COSC 341
Theory of Computing
Lecture 18
Propositional logic and satisfiability

Stephen Cranefield
stephen.cranefield@otago.ac.nz

Lecture slides (mostly) by Michael Albert

Keywords: Propositional logic, satisfiability

Aim: to find our first $\mathbf{NP}$-complete problem

- Finding our first $\mathbf{NP}$-complete problem is daunting as we must be able to reduce *all* problems in $\mathbf{NP}$ to it.
- But once we find our first such problem *prob* then any problem we can reduce *prob* to is also $\mathbf{NP}$-complete.
- Gradually we can build up a catalogue of $\mathbf{NP}$-complete problems.
- Spoiler: that first $\mathbf{NP}$-complete problem is going to be satisfiabilty in propositional logic.

Overview of propositional logic (1)

- VAR is a finite (but large) set of Boolean variable symbols.
- $\wedge$ and $\vee$ are binary operators and $\neg$ is a unary operator.
- We define a context-free grammar of logical *formulas* over the variables:

$$E \to V$$

$$E \to E \wedge E$$

$$E \to E \vee E$$

$$E \to \neg E$$

$$V \to v \quad \text{(for any } v \in \mathrm{VAR}\text{)}$$

- What does $a \wedge b \vee c$ mean?
- Oops! It is ambiguous. We need to add brackets as non-terminals in our grammar:

$$E \to (V)$$

$$E \to (E) \wedge (E)$$

$$\ldots$$

Overview of propositional logic (2)

- A truth assignment is a map $T : \mathrm{VAR} \to \{\mathsf{t}, \mathsf{f}\}$.
- Given a truth assignment, we define the evaluation of a formula recursively:

$$
\begin{aligned}
\mathrm{ev}(v, T) &= T(v) \\
\mathrm{ev}(\neg(E), T) &= !\,\mathrm{ev}(E, T) \;\; \text{where } !\mathsf{t} = \mathsf{f} \text{ and } !\mathsf{f} = \mathsf{t} \\
\mathrm{ev}((E_1) \wedge (E_2), T) &= \begin{cases} \mathsf{t} & \text{if both } \mathrm{ev}(E_1, T) \text{ and } \mathrm{ev}(E_2, T) \text{ are } \mathsf{t} \\ \mathsf{f} & \text{otherwise} \end{cases} \\
\mathrm{ev}((E_1) \vee (E_2), T) &= \begin{cases} \mathsf{f} & \text{if both } \mathrm{ev}(E_1, T) \text{ and } \mathrm{ev}(E_2, T) \text{ are } \mathsf{f} \\ \mathsf{t} & \text{otherwise} \end{cases}
\end{aligned}
$$

- Two formulas are *logically equivalent* if they have the same evaluation for *every* truth assignment, e.g. $a \wedge (b \vee c)$ and $(a \wedge b) \vee (a \wedge c)$.

Conjunctive normal form and satisfiability

- A literal (in logic) is a atomic formula or its negation. In propositional logic, the atoms are variables, so the literals are variables and negated variables.
- A clause is a disjunction of literals, e.g.:

$$x_0 \vee x_2 \vee \neg\, x_6$$

- A formula in conjunctive normal form (CNF) is a conjunction of clauses, e.g.:

$$(x_0 \vee x_2 \vee \neg x_6) \wedge (x_1 \vee \neg x_3 \vee x_4 \vee \neg x_6) \wedge (x_1 \vee x_3 \vee x_5) \wedge (x_0 \vee \neg x_7)$$

- A formula is satisfiable if it evaluates to `t` for *some* truth assignment T. T is called a satisfying assignment.
- The satisfiability problem (slide 8) can take a formula in CNF because:

Theorem

Every formula in propositional logic is logically equivalent to one in CNF.

Proof.

- Construct the truth table for E
- For each row with value `f` create a clause denying that truth assignment.
- Take the conjunction of those clauses.

$\square$

Proof illustration

Consider $(q \wedge \neg p \wedge \neg r) \vee (p \wedge r)$.

p	q	r	$(q \wedge \neg p \wedge \neg r) \vee (p \wedge r)$	Created clause
f	f	f	f	$p \vee q \vee r$
f	f	t	f	$p \vee q \vee \neg r$
f	t	f	t	
f	t	t	f	$p \vee \neg q \vee \neg r$
t	f	f	f	$\neg p \vee q \vee r$
t	f	t	t	
t	t	f	f	$\neg p \vee \neg q \vee r$
t	t	t	t	

CNF: $(p \vee q \vee r) \wedge (p \vee q \vee \neg r) \wedge (p \vee \neg q \vee \neg r) \wedge (\neg p \vee q \vee r) \wedge (\neg p \vee \neg q \vee r)$

From https://math.stackexchange.com/questions/3549712/how-to-compute-cnf-from-truth-table

(Potential) exponential blow-up

Consider the formula $(x_1 \wedge y_1) \vee (x_2 \wedge y_2) \vee .... \vee (x_n \wedge y_n)$.

- This has size linear in n (ignoring the $\log n$ factor for storing variable names).
- Converting this to CNF produces a formula with 2^n clauses:

$$
\begin{aligned}
&(x_1 \vee x_2 \vee \ldots \vee x_n) \\
\wedge\ &(y_1 \vee x_2 \vee \ldots \vee x_n) \\
\wedge\ &(x_1 \vee y_2 \vee \ldots \vee x_n) \\
\wedge\ &(y_1 \vee y_2 \vee \ldots \vee x_n) \\
&\ldots \\
\wedge\ &(y_1 \vee y_2 \vee \ldots \vee y_n)
\end{aligned}
$$

- Each clause contains either x_i or y_i (or its negation) for each i.
- Can we do better? By counting the number of falsifying assignments needed we can show that there must be at least $(1.5)^n$ clauses.

Satisfiability

Satisfiability or SAT
Instance: A formula in CNF over a set of variables V.
Problem: Does the formula have a satisfying assignment?

- The space requirement to describe an instance of SAT containing k clauses over n variables is $O(kn \log n)$ (see Notes 16).
- The non-deterministic "guess and check" procedure to verify an instance requires time at most quadratic in the size of the input (see Notes 16).
- So SAT is in $\mathbf{NP}$.

What makes formulas in CNF easy or hard to satisfy?

- A clause with k variables has 2^k truth assignment over those variables.
- How many of these truth assignments make the clause false?
- Only 1. Clauses are hard to falsify. Each disjunct must be false.
- Adding more variables to a clause makes it even harder to falsify / easier to satisfy.
- Adding more clauses to a formula in CNF adds more falsifying truth assignments.
- The greater the number of clauses in a formula in CNF and the shorter those clauses, the more likely the formula is to be unsatisfiable.

SAT is $\mathbf{NP}$-complete (trailer)

How can we possibly reduce *every* $\mathbf{NP}$ problem to SAT?

Let a TM, M solve some problem in $\mathbf{NP}$ (in polynomial time). As it runs we could imagine taking a snapshot of its configuration at each time step.

- Describing such a snapshot using Boolean variables is pretty straightforward.
- But:
  - How do we ensure the the snapshots change correctly from one time step to the next?
  - How do we limit the size given that the TM has an infinite tape?
  - How do we create a formula that is satisfied exactly when M accepts its input?